ENSF 594 – Principles of Software Development II

Lab Assignment 1

Zach Frena

Question 1:

The Big O complexity is $O(n^2)$. The highest order of T(n) is n^2, and we can ignore the lower order terms and coefficients.

Question 2:

From most to least efficient:

1. Constant
2. Logarithmic
3. Linear
4. Quadratic
5. Cubic
6. Exponential

Question 3:

The Big O running time is $O(n^2)$ because there is a nested for-loop the code must jump through. Technically this is due to the "product rule": if a process is repeated for each "n" of another process, then O is the product of the O's of each process.

Question 4:

The Big O running time is $2 * O(n) = O(n)$ because there are only 2 subsequent for-loops the code must step through, and no nested loops.

Question 5:

The Big O running time is $O(logn)$ because there are only 2 subsequent for-loops the code must jump through, and no nested loops. The log portion comes from the division of the counter within the loop.

Question 6:

```
int test=0;

for (int i = 0; i<n; i++){

        for (int j = 0; j<n; j++){

                for (int k =0;k>n;k++){

                        test=test+i*j*k;

                        }

                }

}
```

Question 7:

7^2 = 49.

Question 8:

The Big O running time is $O(1)$ because there are no loops, just a single if-else statement.

Question 9:

The Big O running time is $O(logn)$ because there is a singular loop which runs the number of times specified by the function's parameters, however it is logarithmic because the loop counter is doubled (i.e. the number of iterations is halved every iteration).

Question 10:

i = 1; // 1 op

sum = 0; // 1 op

while (i <= n) { // 1 op x (n+1)

i=i+1; // 2 ops x n

sum = sum + i; // 2 ops x n

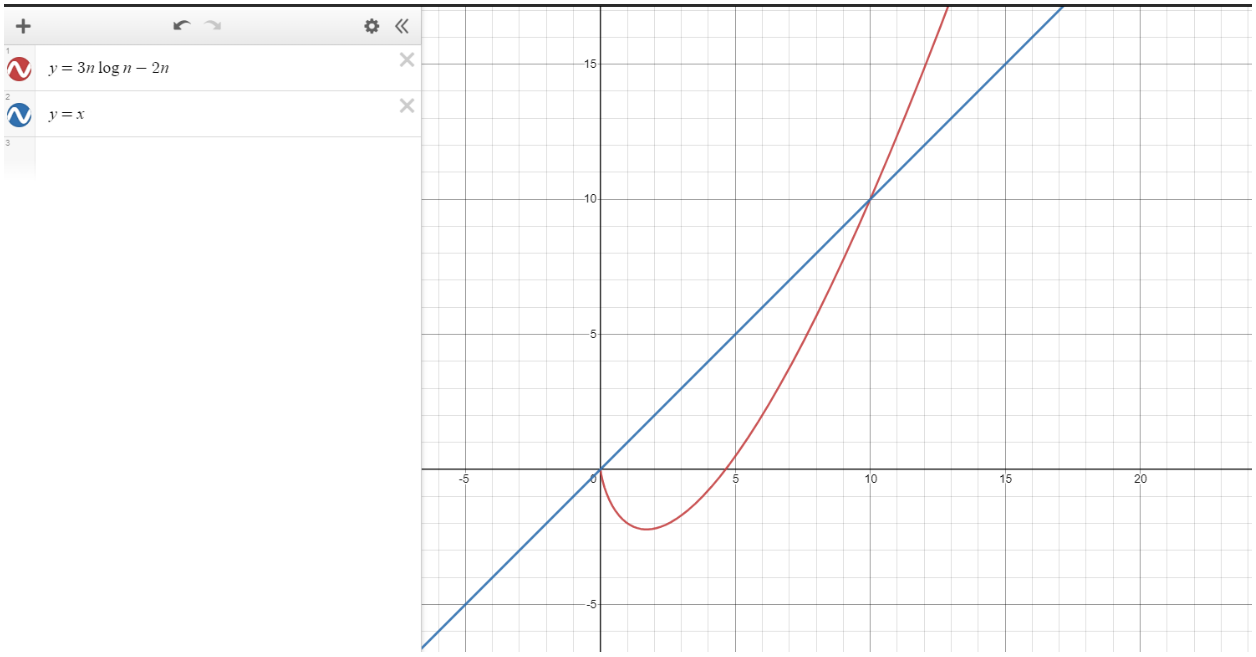
t(n) = 1 + 1 + 1n + 1 + 2n + 2n

t(n) = 3 + 5n

From the complexity calculation above, we can say that the Big O running time is $O(n)$ because there is only 1 loops the code must jump through, and no nested loops. We can confirm this through step-counting because the highest power of the resulting t(n) function is n^1 (i.e. the order of t(n) is 1) and all lower terms are ignored.

Question 11:

The *tightest* lower bound of the function is $O(n)$, because as n trends towards infinity we can see that the algorithm run-time grows at a rate higher than y=x after a certain n_0 (at 10 in this case).


$y = 3n \log n - 2n$
$y = x$